Introduction à la programmation de microcontrôleurs PIC16F84 sur une Linuxette

Jean-Marc LICHTLE

3 novembre 2002

Table des matières

Table des figures

1 Introduction

1.1 Les microcontrôleurs

Les microcontrôleurs envahissent notre environnement sans que nous le sachions. Ces petits composants se retrouvent de plus en plus dans tous les matériels que nous utilisons quotidiennement, machine à laver, mulot (souris), ordinateur, téléviseur. Dotés d'une logique programmée ils sont capables de réagir à l'environnement un peu à la manière d'automates programmables. Mais leurs propriétés ne se limitent pas à offrir un certain nombre d'entrées sorties logiques. Ils sont parfois dotés de fonctions supplémentaires telles que convertisseurs analogiques numériques, horloges temps réel, comptage rapide etc. L'intérêt pour ces composants est directement fonction de leur prix. Sachez que vous pouvez, pour moins de 10 Euros, faire l'acquisition d'un μc tout à fait intéressant.
Plusieurs fondeurs se partagent ce marché, citons INTEL, MOTOROLA, AMTEL, ZILOG, PHILIPS et enfin MICROCHIP qui nous intéresse ici.

1.2 Les microcontrôleurs PIC16F84

MICROCHIP est l'un des grands pour ce qui concerne la fourniture de μc. La gamme des produits proposés se décline en trois grandes gammes, le PIC16F84 étant l'un des représentant de la gamme moyenne. Le stockage des informations, essentiellement le programme, peut se faire de 3 manières, en ROM, EEPROM et mémoire flash. La technologie employée se reflète dans la désignation du composant. Dans notre cas le F de PIC16F84 signifie flash. Vous trouverez couramment dans des appareils grand public des versions CR ce qui signifie ROM.

Dans l'emploi et la mise en oeuvre de μc il est important de prêter attention aux niveaux électriques. Les μc de MICROCHIP utilisent des niveaux TTL 0-5V. Il faut donc veiller lors de la réalisation d'un montage, à ne jamais appliquer une tension supérieure aux pattes ce qui détruirait immédiatement le circuit. De même les sorties ne peuvent débiter plus de 20 mA. Attention donc aux courts-circuits !. Il y a toutefois une exception, lors de la programmation du composant l'une des pattes désignée par MCLR pour "master clear" doit être portée à un niveau compris entre 12V et 14 V.
Vous trouverez sur Internet une masse de renseignements sur ces petites bêtes. Je résumerais ici la "data sheet" du 16F84 en disant qu'il s'agit d'une puce 18 pattes programmable et dont les caractéristiques essentielles sont les suivantes :

- Mémoire flash 1k mots de 14 bits
- 68 octets de RAM pour le stockage des variables non rémanantes
- 64 octets d'EEPROM ce qui permet de stocker des variables rémanantes, réglages, paramètrages etc..
- Architecture RISC (signifie jeu d'instruction réduit)
- 13 ports E/S
- Fréquence d'horloge 4 MHz pour les plus courants et jusqu'à 20 MHz selon le modèle
- Watchdog etc..

Le PIC16F84 fait partie d'une très grande famille de μc produits par MICROCHIP et qui compte des dizaines de variantes différentes, certaines présentant des particularités passionnantes, convertisseur A/N 8, 10 ou 12 bits, communication série ou I2C etc.

1.3 Objectif de la présente étude

L'objectif de la présente est de décrire les outils strictement nécessaires à la programmation de μc avec un PC tournant sous LINUX.

1.3.1 L'approche classique

Il est relativement simple de faire l'acquisition d'un "starter kit" chez l'un des revendeurs de MICROCHIP et de se lancer dans la programmation sous Windows (PIC START 272 Euros TTC, mini programmeur PIC01 59 Euros TTC[1]). Ce faisant vous allez certainement arriver à un résultat rapide mais vous dépenserez aussi pas mal d'argent, ne serait-ce que pour la platine qui servira à la programmation du PIC. De plus, sorti de l'univers douillet de Windows, vous ne serez pas tellement plus avancé.
Par ailleurs cette méthode classique impose de déplacer le PIC du circuit d'utilisation au programmateur et retour à chaque essai d'une nouvelle version du soft (et croyez-moi, au début on en fait des essais...). Conséquence immédiate : il est vivement recommandé d'utiliser des supports ZIF (force d'insertion nulle) pour monter les PIC ce qui sale encore la note pour le débutant.

[1] Les prix indiqués sont ceux du catalogue 2002-2003 de GO-TRONIC

1.3.2 L'approche proposée

J'ai donc choisi volontairement une approche totalement différente :

- Utilisation d'un PC sous LINUX. Dans mon application j'ai remis en service un vieux P75 qur lequel j'ai installé une RED HAT 7.0, pas vraiment le dernier cris mais une distribution solide et fiable et qui se contente sans gros problème d'un petit espace disque
- Programmation directement sur la platine d'utilisation, sans déplacer le PIC du programmeur à l'utilisation et retour. Ce concept est parfois appellé programmation in-situ, ICP ou ICSP pour les anglophones (pour In Circuit Programming ou In Circuit Serial Programming)
- Circuiterie utilisant le strict minimum de composants selon un principe très largement décrit par David TAIT, "Quick and dirty" qui signifie rapide et sale ! Accessoirement l'investissemment restera très faible.

Je n'ai donc à aucune étape choisi la facilité. Mais le résultat justifie largement les choix initiaux. Une fois passée la phase d'apprentissage et d'essais vous serez en mesure de concevoir des circuits de commande efficaces à base de μc dont vous pourrez modifier la programmation sur place, sans toucher aux composants et avec n'importe quel PC, qu'il s'agisse de votre chère Linuxette mais aussi de n'importe quelle trapanelle tournant sous Windows ou DOS ! Le logiciel de transfert de données existe en effet pour toutes ces plateformes, dans des versions certes légèrement différentes mais relativement compatibles. Pour ce qui est de l'assembleur je n'ai pas vérifié si on le trouvait pour DOS. En cas de besoin les aficionados de Bill Gates devront donc se rabattre sur Windows !

1.3.3 Les limitations

Inutile de faire croire que tout est possible et que, moyennant quelques astuces, il devient possible de faire aussi bien et aussi confortable qu'avec un starter kit. Une telle affirmation serait une tromperie. Le dispositif proposé permet simplement d'assembler un programme et de le transférer dans le PIC. Il ne peut pas :

- Lire le programme contenu dans le PIC, par exemple pour vérifier que le transfert a été fait correctement
- Assister l'utilisateur lors du débogage en lisant la valeur des variables, compteurs etc.
- Transférer des paramètres de réglage sous forme de valeurs enregistrées dans la zone EEPROM

Vous serez donc seul face à la bête, sans la moindre aide si celle-ci ne veut pas faire ce que vous voulez !

Ces limitations n'enlèvent toutefois rien à la validité du concept qui est de proposer le système mininaliste qui permette de débuter et/ou de se dépanner si d'aventure on se trouve un jour confronté à un problème de PIC sans avoir sous la main le matériel qui va bien.

FIG. 1 – le circuit en configuration utilisation....

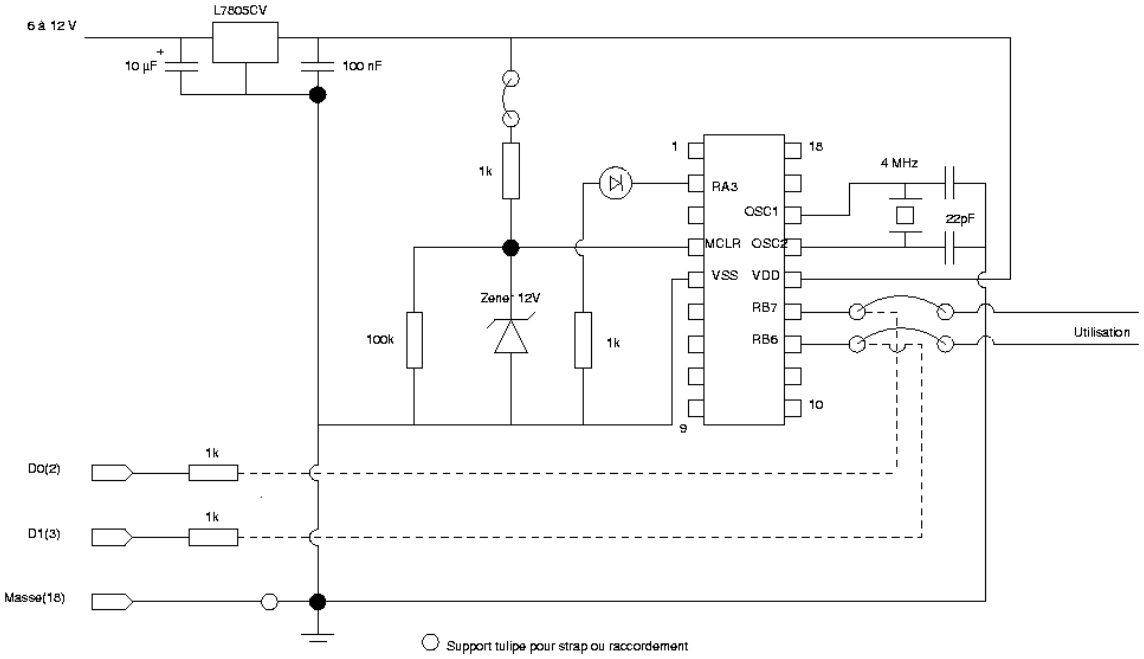
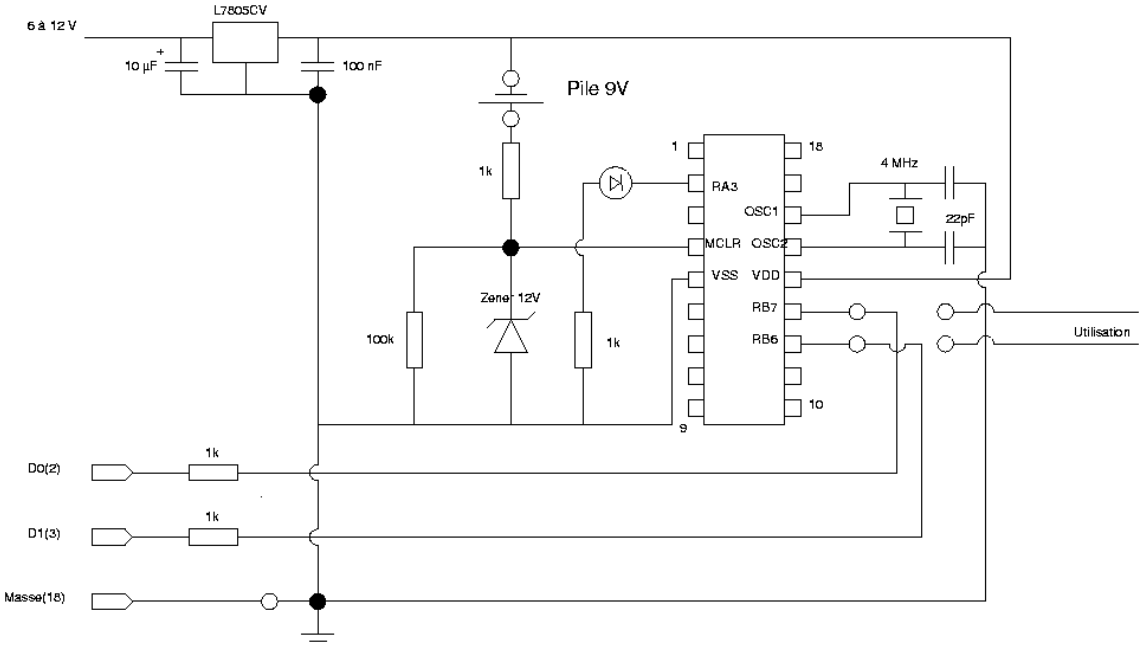


FIG. 2 – le même en configuration programmation.



2 Le circuit

Vous constaterez aisément que le circuit est d'une simplicité biblique. Cette simplicité ne dispense toutefois pas de quelques commentaires.
J'ai dessiné deux variantes du schéma, l'une relative à la configuration utilisée pour l'exploitation, l'autre pour la programmation. Vous jouerez utilement au jeu des erreurs pour trouver les différences.

2.1 Quartz

Pour fonctionner le PIC16F84 a besoin d'un quartz qui va lui permettre de définir une fréquence d'horloge. Les 16F84 courants supportent une fréquence de 4MHz ce qui est déjà une belle performance mais il existe des F1 qui roulent à 20 MHz ! Un conseil : il existe des quartz de 4.000 MHz et des quartz de 4.096 MHz. Pour ma part je préfère la deuxième fréquence qui permet après une division judicieuse ($4096=2^12$) de faire un chronomètre ou toute autre application nécessitant une base de temps.
Le quartz est complété par deux condensateurs 22 pF raccordés à la masse.

2.2 Alimentation

Le circuit est alimenté par un régulateur 5V positif selon une disposition tout à fait classique. Certains auteurs préconisent de monter un condensateur de 0.1 μF céramique à proximité immédiate de la broche VDD du circuit PIC16F84.
Une branche de l'alimentation est raccordée à la broche MCLR du circuit, broche qui commande le reset général lorsque sa tension tombe à zéro. Sur le circuit tel que dessiné cette broche est raccordée à la tension d'alimentation via une résistance de 1k dont nous verrons l'intérêt dans un instant. Un strap permet de connecter et de déconnecter MCLR.

2.3 Tension de programmation

Pour programmer le circuit il est nécessaire de faire passer la tension de la broche MCLR à 12V (12 à 14V suivant la data sheet). C'est là qu'interviennent les composants qui constituent la branche d'alimentation de MCLR. Compte tenu du fait que la programmation est une opération qui ne nécessite que quelques secondes j'ai choisi de fournir cette tension en insérant une pile 9V à la place du strap qui est figuré en haut du schéma. Ce faisant la tension totale disponible passe à 14V, tension réduite ensuite à l'entrée de MCLR par la combinaison résistance 1k et zener 12V. C'est simple et très rapide à mettre en oeuvre.

- Pour mettre le circuit en marche normale mettre le strap comme figuré sur le dessin du circuit en configuration utilisation
- Pour provoquer un reset retirer le strap, la résistance de 100 k montée en parallèle avec la zener tire alors le potentiel de MCLR vers la masse
- Pour monter à la tension de programmation remplacer le strap par une pile 9V, le + étant dirigé vers MCLR

Les barrettes sécables à contacts tulipe sont d'excellentes embases de straps. Avec un petit anneau de thermorétractable de couleur on donne une petite touche professionnelle qui démontrera à quel point le travail a été bien pensé....

2.4 Circuit de programmation

Le PIC16F84 se programme en appliquant un signal d'horloge sur la broche RB6 et les informations binaires sérialisées sur la broche RB7. Chacune des informations qui transite sur la broche 7 est validée à la retombée du signal d'horloge sur la broche 6. Les niveaux électriques étant des niveaux TTL le plus simple est donc d'employer le port parallèle pour fournir les niveaux électriques souhaités. Le logiciel que nous utiliserons plus loin fournit l'horloge sur la broche 3 du connecteur DB25 et les informations sur la broche 2. Des résistances de 1k qui pourront être montées directement dans le capot du connecteur, limitent le courant en cas de fausse manoeuvre, court-circuit ou autre aléas. Là aussi les manchons thermorétractables seront d'un grand secours pour isoler tout ce petit monde. Le port parallèle d'un PC est un composant qui ne supporte absolument pas de fournir un courant de plus de 20 mA. C'est le motif pour lequel je choisis de le protéger au plus près, directement dans le connecteur. Attention : Vous pourriez être tentés d'utiliser des liaisons très longues pour vous simplifier l'existence. Gardez vous en bien. Il vaut bien mieux acheter un cordon type rallonge de cordon d'imprimante (DB25/DB25) lequel sera correctement blindé plutôt que d'allonger les fils de raccordements terminaux. Autre possibilité : Utiliser un connecteur Centronics femelle qui se branchera directement sur un cordon d'imprimante. Plus difficile à trouver à mon avis, par ailleurs il vous faudra vérifier les numéros de broches, les valeurs que je donne sont relatives à des connecteurs DB25. Là aussi les supports tulipes feront d'excellents connecteurs pour le circuit de programmation.

2.5 Utilisation des broches RB6 et RB7

Ces broches sont utilisées pour la programmation du circuit. Celà ne signifie toutefois pas qu'elles soient inutilisables par la suite, une fois le programme en place. La seule contrainte est que ces broches doivent être libres de potentiel au moment de la programmation ce qui signifie qu'elle doivent être séparées du circuit d'application. Là aussi des supports tulipes coupés dans des barettes sécables et des straps permettent de réaliser à moindre frais une connexion pour la programmation ou une connexion pour l'utilisation.

2.6 La branche de test

Tout ça est bien beau mais à quoi celà va-t-il servir me direz vous ? Et bien pour un premier essai nous pourrons nous contenter d'un objectif humble mais tellement utile, faire clignoter une LED. Cette LED sera simplement connectée à la broche 2 du circuit (RA3), le courant débité sera limité par une résistance de 1k montée en série (décidément on consomme beaucoup de 1k dans cette application !).

3 Notre premier programme

3.1 Utilitaire d'assemblage

Le PIC16F84 peut se programmer dans divers languages mais la langue de base, celle qui est la plus performante, est l'assembleur. Il s'agit d'un language très proche du langue machine ce qui lui confère une efficacité inégalée. Inconvénient, son utilisation va vous obliger à penser comme un ordinateur ce qui n'est pas évident à priori.
La première opération consiste donc à trouver un assembleur qui tourne sous LINUX, celui qui est mis en ligne par MICROCHIP, MPASM, ne convenant à l'évidence pas puisque tournant sous Windows ! Vous pourrez trouver votre bonheur sur Internet, pour ma part je me suis arrêté au choix d'un package nommé "gputils". Présenté au format .rpm il est très simple a mettre en place. L'auteur de gputils, James BOWMAN, présente son produit comme devant, à terme, reproduire toutes les fonctionnalités des logiciels de MICROCHIP. Pour l'instant le logiciel est en version alpha mais l'essentiel est présent puisque le compilateur gpasm est opérationnel. Le paquetage gputils (version 0.10.3) comporte par ailleurs :

- gpdasm, un desassembleur qui permet d'analyser du code compilé si on en possède pas le source
- gpvc, un visualisateur de code

Vous trouverez toute la documentation nécessaire sur les prestations offertes par gputils dans la documentation livrée avec le paquetage et qui s'installe classiquement dans /usr/share/doc/gputils...
Le paquetage gputils peut être récupéré sur

`http://www.rpmfind.net`

Une autre possibilité est de visiter

`http://gputils.sourceforge.net/.`

3.2 Notre premier code source

Les lignes qui suivent contiennent le code source que nous allons utiliser pour notre premier essai. Comme exposé ci-dessus le but que nous nous sommes fixé est très simple : Faire clignoter une LED alimentée par la patte 2 du circuit. Le code est des plus simples, sa compréhension ne devrait pas poser de problème insurmontable pour un débutant qui ferait l'effort d'étudier les rudiments du langage assembleur. Ce code peut se retrouver dans diverses variantes sur Internet, certaines n'allumant qu'une LED, d'autres réalisant un chenillard de 8 LED. La comparaison des codes sera une source d'informations très intéressantes.

```
;*****************************************************************
;                           testled1.asm
;
; Ce code provoque un clignotement d'une led raccordée à la broche 2 du PIC
; Il est largement inspiré d'une programme rédigé par fpederse
; Ma contribution a consisté à élaguer tout ce qui n'est pas strictement
```

```
; nécessaire et à mettre des commentaires dans la langue de MOLIERE.
; J'ai laissé de code des tempos de 10, 30 et 60 sec. de telle sorte à
; permettre des essais à des valeurs différentes afin de vérifier que les
; transferts de programmes successifs se déroulent normalement.
;*****************************************************************************

  LIST    P=16F84;f=inhx8m

_CP_OFF         equ     H'3FFF'  ; Code protect hors service
_PWRTE_ON       equ     H'3FFF'  ; Power on timer en service
_WDT_OFF        equ     H'3FFB'  ; Watch dog timer hors service
_XT_OSC         equ     H'3FFD'  ; Crystal oscillator (quartz)
  __CONFIG       _CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC
; Combinaison des paramètres

;-----------------------------------------------------------------------
; Directives de configuration
;-----------------------------------------------------------------------

count1  equ     0C              ; Le premier compteur
count2  equ     0D              ; Le second
portb   equ     06              ; Registre de port B
porta   equ     05              ; Registre du port A
status  equ     03              ; Le registre status
rp0     equ     05              ; Le bit 5 == sélection page de mémoire
trisa   equ     85H             ; Reg. de réglage en entrée ou sortie du port A


;-----------------------------------------------------------------------
; Programme
;-----------------------------------------------------------------------

  org   0          ; origine
  clrf  porta      ; met toutes les sorties du port A à zéro
  bsf   status,rp0 ; sél. page mémoire 1 pour accès à trisa
  clrf  trisa      ; reset trisa, port A est défini en sortie
  bcf   status,rp0 ; sélection page mémoire 0

;-----------------------------------------------------------------------
; Code modifiable pour essais successifs
;-----------------------------------------------------------------------
start
  bsf   porta,3    ; Fixe à 1 le bit 3 du port A -> 5V sur broche 2
```

```
  call  wait_sec_5 ; Tempo LED allumée
  bcf   porta,3    ; Fixe à 0 le bit 3 du port A -> 0V sur broche 2
  call  wait_sec_5 ; Tempo LED éteinte
  goto  start      ; Boucle sur étiquette start

;-------------------------------------------------------------------------
; Sous programmes de temporisation
;-------------------------------------------------------------------------
wait_min           ; tempo 1 min (non utilisée ici)
  call  wait_sec_30
  call  wait_sec_30
  return
wait_sec_30        ; tempo 30 sec (non utilisée ici)
  call  wait_sec_10
  call  wait_sec_10
  call  wait_sec_10
  return
wait_sec_10        ; tempo 10 sec (non utilisée ici)
  call  wait_sec_5
  call  wait_sec_5
  return
wait_sec_5         ; tempo 5 sec
  call  wait_sec
  call  wait_sec
  call  wait_sec
  call  wait_sec
  call  wait_sec
  return
wait_sec           ; tempo 1 sec
  call  wait1
  call  wait1
  return
wait1              ; 4 tempos élémentaires
  call  wait0
  call  wait0
  call  wait0
  call  wait0
  return
wait0              ; Boucles réalisant une tempo élémentaire
   movlw  .200     ; Charge count1 avec valeur décimale 200
   movwf  count1
d1 movlw  .200     ; Charge count2 avec valeur décimale 200
   movwf  count2
```

```
d2 decfsz count2,1 ; décremente et saute ligne suiv. si zero
   goto   d2       ; si différent de zero
   decfsz count1,1 ; décremente count1 si count2 == zero
   goto   d1       ; boucle interne si count2 <> zero
   return

end
```

3.3 Commentaires sur ce code

3.3.1 Indentations

Le langage assembleur dicte certaines règles, parmi celles-ci le respect d'un certain formalisme dans la mise en forme du code source :

- Les déclarations de variables sont cadrées à gauche (tous les xxx equ xxx)
- Les directives LIST ou __CONFIG sont précédées d'un blanc, tabulation ou espaces(s)
- Les étiquettes sont cadrées à gauche
- Les lignes de code sont, comme les directives CONFIG ou LIST, précédées d'un blanc
- Le cadrage des commentaires, précédés d'un ; est indifférent

3.3.2 La directive CONFIG

J'ai laissé en entête les lignes qui définissent les paramètres combinés par une fonction ET et qui définissent la configuration souhaitée. Ces quelques lignes auraient aussi bien pu être remplacées par une simple ligne __ CONFIG H'3FF1'. La valeur hexa '3FF1'donne en binaire 0011 1111 1111 0001. Vous vous reporterez utilement à la documentation existante pour retrouver la signification des différents bits, par exemple sur

`http://www.sq-1.com/config.html`

En résumé, de droite à gauche :

- bits 0 et 1 : 01, utilisation d'un oscillateur XT c'est à dire quartz jusqu'à 4 MHz
- bit 2 : 0, Watch dog timer désactivé, le chien de garde qui surveille le temps de cycle ne sera donc pas activé
- bit 3 : 0, Power up timer activé, à la mise sous tension il va s'écouler une très légère tempo avant que la scrutation du programme démarre
- bits 4 à 13 : suite de 1, Code protect désactivé

3.3.3 Mnémoniques de configuration

Dans le même ordre d'idée j'ai laissé la liste détaillée des variables et mnémoniques. L'autre technique aurait été de faire appel à une directive d'inclusion d'un fichier contenant toutes ces nmémoniques (et bien d'autres) avec leurs valeurs respectives. On aurait ainsi trouvé à la place de la liste une simple ligne :

```
#include <p16f84.inc>
```

C'est évidemment plus sobre, l'inconvénient est toutefois que dans la suite du code les différents appels à des variables contenues dans le fichier inclus sont moins 'transparents'. Le débutant aura du mal à faire le lien entre l'appel à une mnémonique et une de ses fameuses variables incluses.

3.3.4 L'entête du code source, initialisation du port A

Encore une fois il ne saurait être question de faire de ce document un cours sur l'assembleur appliqué à la programmation des PIC. Vous pourrez utilement vous reporter à ce sujet à un document de fond "La programmation des PICs" de BIGONOFF, par exemple sur

`http://www.abcelectronique.com/bigonoff/`

Le travail réalisé par le code d'entête est assez classique, on fixe l'origine du code dans la mémoire du PIC (peut-être pas nécessaire puisque 0 est l'adresse par défaut), puis on fait un reset des sorties du port A, après quoi on défini les 5 broches du port A comme autant de sorties (RA0 à RA4). Cette manipulation ne peut se faire qu'en sélectionnant la page 1 de la mémoire, le registre trisa n'étant pas accessible depuis la page 0.

3.3.5 Le corps du programme

Le programme à proprement parler n'est constitué que de 6 lignes, une étiquette qui servira à boucler, 4 lignes qui affectent successivement les valeurs 1 et 0 à la pin 2 (RA3) du PIC en laissant passer une tempo entre chaque changement, après quoi la dernière ligne reboucle sur l'étiquette définie plus haut. On ne peut plus simple donc.

3.3.6 Les temporisations

Vous n'aurez aucun mal à démonter le mécanisme des tempos de 60, 30, 10, 5 sec etc. Elle s'obtiennent par répétition d'une tempo de base repérée par l'étiquette wait0.
L'analyse de cette dernière est intéressante. Les deux premières lignes qui suivent l'étiquette ne sont parcourues qu'une seule fois. Elles initialisent le compteur count1 à 200 (valeur décimale ce qui explique la notation curieuse .200). Suivent deux boucles imbriquées, la boucle externe d1 - goto d1 et la boucle interne d2 - goto d2. Le jeu consiste ici à décrémenter les compteurs et à boucler gentiment tant que ceux-ci ne sont pas tombés à zéro. Le passage par zéro du compteur interne (d2) décrémente d1 d'une unité et réinitialise d2 à 200. La mise à zéro de d1 termine la temporisation. Les deux mécanismes sont basés sur des opérateurs decfsz, acronyme qui signifie, en bon français, décrémente la variable et saute si zéro ! En clair la variable nommée, ici count1 ou count2 est décrémentée, la nouvelle valeur est rangée dans la variable (d'ou le ,1). Si cette nouvelle variable est différente de zéro on exécute la ligne de code suivante, sinon on saute une ligne plus loin.
La valeur de cette temporisation est assez facile à déterminer si on garde présent à l'esprit que les lignes de code s'exécutent à raison d'une ligne par cycle d'horloge sauf pour les branchements

qui en nécessitent deux. La boucle interne nécessite donc 3 cycles. Parcourue 200 fois elle consomme 600 cycles. Chaque boucle externe nécessite 5 cycles, plus les 600 cycles de la boucle interne, le tout multiplié par les 200 boucles à parcourir. Il vient donc très en gros 120000 cycles. Sur un pic raccordé à un quartz cadencé à 4 MHz la fréquence interne est de 1 MHz (1/4). La tempo élémentaire sera achevée après 0.12 sec. Quatre tempos wait0 à suivre dureront donc environ 1/2 sec (wait1), deux wait1 dureront 1 sec etc...
Deux remarques :
Le calcul effectué ici est taillé à la hache. Si vous souhaitez déterminer avec précision la durée de la tempo il faudra fignoler un peu (les boucles sont elles décrites 200 ou 199 fois ?). Pour ma part je garde un mauvais souvenir des problèmes du genre nombre d'arbres et nombre d'intervals.
La temporisation par boucles successives est une horreur puisqu'on consomme de la puissance pour faire passer le temps. Il existe d'autres techniques bien plus judicieuses et qui utilisent, par exemple, le chien de garde. Vous n'aurez aucun mal à trouver le source d'un logiciel utilisant cette méthode, cherchez par exemple count.asm sur Internet.

4 L'assemblage, utilisation de gpasm

Le paquetage gputils fournit, comme nous l'avons vu brièvement plus haut, le logiciel gpasm qui permet d'assembler le programme, c'est à dire transformer le code source rédigé en mnémoniques compréhensibles par un être humain (entraîné !) en code machine. Comme d'habitude l'appel de gpasm avec l'extension -? ou -h provoque l'affichage d'un écran d'aide reproduit ci-dessous.

```
Usage: gpasm [options] file
Options: [defaults in brackets after descriptions]
  -a FMT, --hex-format FMT       Select hex file format. [inhx8m]
  -c, --case                     Case insensitive.
  -D SYM=VAL, --define SYM=VAL   Define SYM with value VAL.
  -e [ON|OFF], --expand [ON|OFF] Macro expansion.
  -h, --help                     Show this usage message.
  -I DIR, --include DIR          Specify include directory.
  -L, --force-list               Ignore nolist directives.
  -l, --list-chips               List supported processors.
  -m, --dump                     Memory dump.
  -n, --dos                      Use DOS newlines in hex file.
  -o FILE, --output FILE         Alternate name of hex file.
  -p PROC, --processor PROC      Select processor.
  -q, --quiet                    Quiet.
  -r RADIX, --radix RADIX        Select radix. [hex]
  -w [0|1|2], --warning [0|1|2]  Set message level. [0]
  -v, --version                  Show version.

Reading header files from /usr/share/gputils/header
```

```
Report bugs to:
<URL:http://gputils.sourceforge.net/>
```

Compte tenu des détails contenus dans le code source proposé les différentes options proposées ne nous servent à rien ! Nous avons en effet fixé le type de processeur à 16F84 et le format de sortie à inhx8m par la directive LIST en début de code. Le format des nombres est hexa par défaut. Il en découle la notation .200 qui apparaît à divers endroits pour forcer la lecture d'un nombre décimal.
L'assemblage se fera donc simplement par la ligne suivante :

```
gpasm testled1.asm
```

à laquelle le compilateur va réagir par l'affichage suivant :

```
tl1.asm:42:Message [302] Register in operand not in bank 0.
Ensure bank bits are correct.
```

Passé le premier instant d'affolement (mais bon sang qu'est ce qui ne va pas dans ce source, jml raconterait-il des bétises ! ?) vous vous rendrez compte que vous êtes devant un simple message d'information et non une bordée d'injure induite par une erreur d'assemblage. Ce message provient de la ligne 42, en clair celle qui efface les bits du registre trisa (clrf trisa). Gpasm vous demande simplement si vous êtes certain d'avoir sélectionné la bonne page mémoire, la page 1 qui permet l'accès au registre trisa. Notez que si vous voulez éviter un raté cardiaque à chaque assemblage vous pouvez sélectionner un niveau de message moins sensible. A parir du niveau 1 (gpasm -w 1 testled1.asm) l'assembleur considère avoir affaire à un gourou et évite de l'incommoder avec des détails mineurs ! Dans quelques mois vous vous permettrez peut-être même le niveau 2. Souvenez vous simplement, arrivé à ce niveau, que vos premiers pas ont été guidés par un certain jml qui a trimé seul pendant des semaines pour mettre au point ce document. Encore tout content du coup de main vous aurez peut être envie de lui envoyer un petit mail sympa. N'hésitez pas, l'adresse est jean-marc.lichtle@gadz.org.
Plus sérieusement l'opération d'assemblage a créé un certain nombre de fichiers :

- testled1.hex, le fichier en code machine qui devra être transféré dans le PIC
- testled1.cod, un fichier visualisable par gpvc et qui contient une masse d'informations sur le code assemblé, sa taille, les variables etc.
- testled1.lst, un fichier texte qui établi (entre autre) une correspondance entre les lignes de code source et leur traduction en hexa

Je vous laisse juge de l'intérêt de ces différents fichiers, nul doute qu'arrivés au niveau gourou vous saurez en faire un savant usage. Dans l'immédiat ce qui nous intéresse est la présence du fichier testled1.hex, confirmation que la compilation a donné un résultat. Le tout est de savoir si ce résultat présente un intérêt, ce sera l'objet du chapitre suivant.

5 Transfert vers le PIC, utilisation de pp

Le transfert vers le PIC va être réalisé au moyen d'un logiciel très rustique avec une interface en mode texte répondant au joli nom de pp, pour PIC Programmer. L'auteur de ce produit est Chris WILSON. Vous trouverez facilement la dernière version du soft sous forme d'archive tar sur Internet, par exemple à l'adresse :

`http://ftp.unicamp.br/pub/systems/Linux/system/network/isode/.`

La décompression de l'archive dans sa version actuelle crée un sous-répertoire ./pp-0.6 dans lequel vous trouverez quelques fichiers donc les incontournables README.1ST, README, mais aussi FILES.TXT qui donne la liste des fichiers reproduite en partie ci-dessous (et traduite) :

- FILES.TXT, la liste des fichiers
- PP.TXT, une brève description du programmeur et des incidences sur le software, principalement les variables d'environnement
- PROGRAM.TXT, mode d'emploi du logiciel de transfert
- PP.PCX, le schéma du programmeur PIC16X8X proposé par Davis TAIT, une version complète assez proche d'un programmeur commercial
- pp, le logiciel de programmation sous LINUX (et aussi PP.EXE pour les utilisateurs de Microsoft)
- SRC.ZIP, le code source en C
- QANDD.PCX, le schéma du prgrammeur quick-and-dirty proposé par David TAIT, dont je me suis inspiré pour cet article
- QANDD.TXT, une brève description du programmeur quick-and-dirty
- TEST.PCX, schéma d'un circuit de test à 4 diodes
- WALK.ASM, un code source pour réaliser un chenillard, la temporisation est obtenue dans ce cas par un débordement du watchdog
- mypp, un fichier executable qui lance pp après avoir fixé la valeur d'un certain nombre de variables d'environnement. A paramètrer selon votre propre configuration, par exemple si vous programmez avec le port lp1 au lieu de lp0. Attention, la numérotation est de style Windows, le permier port a donc le numéro 1 et non 0 comme avec LINUX

Note : Les fichiers graphiques au format .PCX peuvent être lus avec The Gimp....

Cette liste est loin d'être complète, je n'ai mentionné que les fichiers principaux. Les gourous trouverons aussi un Makefile et d'autres friandises.

Une autre possibilité pour se procurer pp est de récupérer un fichier nommé linuxpp.zip. Celui peut être téléchargé sur :

`http://www.thepicarchive.cwc.net/dtpa/links.html`

Il suffit alors de le copier dans le sous répertoire qui vous sert à vos essais de PIC et de suivre une procédure tout à fait classique résumée ci-dessous (et décrite dans Readme.now) :

```
[jml@jml linuxpp]$ unzip linuxpp.zip
Archive:  linuxpp.zip
  inflating: Readme.now
```

```
 extracting: linuxsrc.zip
  inflating: mypp.sh
[jml@jml linuxpp]$ unzip linuxsrc.zip
Archive:  linuxsrc.zip
  inflating: all-patches
  inflating: Makefile
  inflating: config.h
  inflating: hex.c
  inflating: hex.h
  inflating: linux.c
  inflating: pp.c
  inflating: pphw.c
  inflating: pphw.h
  inflating: timer.c
  inflating: timer.h
[jml@jml linuxpp]$ make
gcc -g -O2 -DLINUX    -c -o pp.o pp.c
pp.c: In function `main':
pp.c:356: warning: return type of `main' is not `int'
gcc -g -O2 -DLINUX    -c -o hex.o hex.c
gcc -g -O2 -DLINUX    -c -o timer.o timer.c
gcc -g -O2 -DLINUX    -c -o pphw.o pphw.c
gcc -g -O2 -DLINUX    -c -o linux.o linux.c
gcc -g -O2 -DLINUX   pp.o hex.o timer.o pphw.o linux.o   -o pp
[jml@jml linuxpp]$ ls
all-patches  hex.h    linux.o       Makefile  pp.c    pphw.o      timer.c
config.h     hex.o    linuxpp.zip   mypp.sh   pphw.c  pp.o        timer.h
hex.c        linux.c  linuxsrc.zip  pp        pphw.h  Readme.now  timer.o
```

Attention, à la différence de la méthode précédente le traitement de linuxpp.zip ne crée pas de sous répertoire ! Veillez à ne pas procéder dans votre répertoire personnel faute de quoi vous auriez du mal à retrouver vos petits plus tard. La différence essentielle entre les deux méthodes réside dans le fait que le traitement de linuxpp.zip donne simplement l'executable sous Linux alors que la méthode précédente vous permet de récupérer aussi pp.exe, l'executable sous DOS (ça peut servir.... si votre linuxette est occupée)

5.1 Les conditions préalables à l'utilisation de pp

Il importe de bien se souvenir (le logiciel saurait vous le rappeler le cas échéans) que l'accès aux ports n'est pas autorisé normalement à l'utilisateur Lambda. Ceux-ci ne sont accessibles qu'à l'administrateur. Une fois n'est pas coutume, nous ferons donc une chose inimaginable en pure logique Linuxienne, nous allons utiliser pp sous compte root ! Cette infraction est toutefois admissible dans la mesure ou nous sommes sur une machine isolée et qui n'a pas d'accès à

Internet ou à un réseau d'entreprise au moment ou elle sert de console de programmation. Une autre solution aurait été de rester sous compte utilisateur et de lancer la commande pp au travers d'une commande su. Cette méthode est toutefois trop lourde lorsqu'on se prépare à une foule d'essais. Elle impose en effet de taper chaque fois le mot de passe administrateur.
Le deuxième aspect important est la nécessité de mettre en place et de paramètrer correctement trois variables d'environnement, PPSETUP, PPDELAY, et PPLPT. Les valeurs correctes sont rappelées dans le fichier script de commande nommé mypp. Pour ma part je préfère, plutôt que d'utiliser ce fichier, taper une fois pour toutes les trois lignes qui suivent :

```
export PPSETUP=3
export PPDELAY=6
export PPLPT=1
```

Vous vous reporterez utilement au contenu de PP.TXT pour la signification de ces varaibles. L'examen du code source de pphw et les nombreux essais effectués démontrent que l'initialisation de PPLPT n'est pas indispensable si vous utilisez le premier port parallèle de votre machine. L'adresse 0x378 est utilisée par défaut si la variable d'environnement n'est pas initialisée. Par contre il est impératif de créer PPSETUP et de l'initialiser à 1 ou 3. Les valeurs 0 et 2 ne conviennent pas. L'explication du motif peut être trouvée dans PP.TXT. Notre hardware minimaliste semble en effet se comporter, du point de vue du soft, comme un programmeur équipé de buffers 7407 non-inverseurs (bien sûr, avec des fils et des résistances on ne risque pas d'inverser quoi que ce soit !). La variable PPDELAY ne semble pas avoir une incidence majeure, du moins sur ma vieille machine. Même non initialisée la programmation fonctionne correctement.

5.2 La syntaxe de pp

La syntaxe est assez simple. Pour effectuer le transfert du fichier testled1.hex il suffit de taper :

```
./pp -n testled1.hex
```

à quoi pp va vous répondre par :

```
Programming hardware not found or is faulty
PIC16F84 Programmer  Version 0.6  Copyright (C) 1994-1998 David Tait.

Insert PIC ... press any key to continue (^C to abort)
```

Répondant à l'invitation qui vous est faite vous taperez alors sur la touche Entrée (par ex.) ce qui aura pour effet de transférer le programme souhaité dans le PIC et d'afficher :

```
Programming ...
Setting config to ---X ...
Finished in 2 secs
```

L'appel à pp via une syntaxe ./pp est lié au fait que pp n'est pas, à priori, dans un des sous-répertoires désigné par la variable d'environnement PATH. Il s'en suit la nécessité de préciser dans quel répertoire est rangé le programme pp, ici le répertoire courant.

La durée du transfert est indicative. Elle va dépendre du volume du fichier. L'option -n est importante. Elle force en effet pp à travailler en aveugle sans essayer de relire ce qu'il a transféré. Cette option est en fait obligatoire faute de quoi pp, se rendant compte que le programmeur n'est pas un modèle classique, va se mettre automatiquement en mode debug.

5.3 Premiers essais

Créez deux versions du logiciel à transférer avec des différences sensibles dans les temporisations, 1+1s et 5+5s ou alors 1+5s et 5+1s. L'objectif est ici de charger l'une des versions puis, à titre d'exercice, de charger l'autre en écrasant la première. Les explications des chapitres précédents devraient vous permettre de créer facilement les deux variantes.
J'ai fait d'innombrables essais, parfois avec des résultats desespérants (le désespoir c'est quand il ne se passe rien et que le doute s'installe : Le PIC est-il foutu ? Ai-je fait une fausse manoeuvre ?) Pour vous éviter les affres de recherches identiques je vous donne donc quelques tuyaux :

- L'alimentation qui débouche sur le régulateur 5 Volts peut rester branchée en permanence. Inutile de chercher à faire un reset en coupant cette alimentation, ça ne sert à rien
- Dans la mesure ou vous n'utilisez pas les sorties RB6 et RB7 dans votre application, ce qui est le cas ici, vous pouvez laisser la connexion avec le port parallèle en place. La littérature MICROCHIP dit que le passage en mode programmation est obtenu lorsque MCLR passe de 0 à 12V alors que RB6 et RB7 sont tous deux à 0V. En fait, vous pourrez le vérifier, ces deux broches restent bien sagement à 0 lorsque le logiciel pp est au repos, à condition bien sûr que PPSETUP soit configuré correctement.

Dès lors une cession courante pourrait se dérouler comme suit :

- Mise sous tension, le PIC se met en marche avec le programme qu'il contient ou reste à l'arrêt s'il est vierge ou effacé
- Retrait du strap qui maintient MCLR à la tension d'alimentation, le PIC s'arrête
- Mise en place de la pile 9V qui délivre la tension de programmation, le PIC passe en mode programmation
- Transfert d'une nouvelle version du soft au moyen de pp en utilisant le syntaxe ci-dessus
- Retrait de la pile 9V et mise en place du strap. Entre ces deux étapes MCLR passe par 0 en provoquant un reset du PIC
- Vous devriez voir repartir le PIC sur son nouveau programme

En respectant cette chonologie vous ne devriez pas avoir de problèmes de transfert.

5.4 Utilisation avancée

Il ne faut pas se mentir, avec le hardware mis en oeuvre il n'est guère question de faire de l'utilisation avancée. Tout au plus peut-on envisager d'effacer complètement le PIC en tapant

./pp -en. Ici aussi l'option "n" est nécessaire pour éviter que pp passe en mode debug. L'effacement se place dans le cheminement décrit ci-dessus à la place de la ligne qui transfert le programme.
Quelques conseils :

- Après un effacement il faut repasser par un mode normal (MCLR alimenté à 5V) avant de transférer un nouveau programme. La succession d'une commande effacement et d'une commande transfert ne semble pas donner le résultat qu'on pourrait être en droit d'attendre
- Dans le même ordre d'idée une syntaxe du style ./pp -en testled1.hex qui tenterait dans la même action d'effacer et de reprogrammer le PIC semble vouée à l'échec, du moins avec le hardware rustique que nous employons.
- Aspect intéressant : on peut reprogrammer en écrasant une ancienne version sans passer par une phase inermédiaire d'effacement.
- Un effacement peut servir à vérifier que le dialogue PIC / logiciel se déroule normalement. Il suffit ensuite de charger un programme réel.

Une aide élémentaire sur pp peut être obtenue avec la syntaxe classique ./pp -?. Attention, même pour vous livrer quelques lignes d'aide pp exige que vous soyez administratur sur votre machine !

```
[root@jml pp-0.6]# ./pp -?
PIC16F84 Programmer  Version 0.6  Copyright (C) 1994-1998 David Tait.

Usage: pp  [ -lxhrwpcdevgosn! ]  hexfile

      Config:   l = LP,   x = XT,     h = HS,      r = RC
                w = WDTE, p = PWRTE,  c = code protect
      Others:   d = dump, e = erase,  v = verify,  g = go
                o = old,  s = silent, n = no read, ! = no wait
      Defaults: RC, /WDTE, /PWRTE, unprotected,
                no erase, stop, new, verbose, read, wait

Bug reports to david.tait@man.ac.uk
Programming hardware not found or is faulty
[root@jml pp-0.6]#
```

Ici aussi vous avez la possibilité, comme lors de la mise en oeuvre de gpasm, de mettre en place les paramètres de configuration. Pour ma part, comme exposé ci-dessus, je préfère figer la config dans le code source. En pratique les seules options qui servent (à mon avis) avec notre mini-hardware sont les options déjà vues plus haut, e pour erase ou n pour no read.
Vous avez enfin la possibilité de lancer pp en mode debug. Là aussi la maigreur du hardware mis en oeuvre ne nous conduira pas bien loin. Comme signalé plus haut pp se met en mode debug si vous entrez une ligne sans préciser l'option -n. Il est donc très rapide d'accéder à cette fonctionnalité. L'affichage obtenu est reproduit ci-dessous, les interlignes correspondent à des frappes de la touche Entrée qui permet de passer à l'étape suivante. Le mode debug vous per-

mettra au minimum de vérifier la continuité de la liaison PC en faisant monter successivement les broches RB6 et RB7 du PIC.

```
[root@jml pp-0.6]# ./pp
Programming hardware not found or is faulty
pp: Hardware not connected
Programming hardware not found or is faulty
Hardware setup: 7406/4066 using LPT1 at 0378 (delay = 6)
Debug mode entered ... (^C to exit)

Remove PIC ...

VPP off, VDD off, RB6 low, RB7 low (input OK) ...

VPP on ...

VDD on ...

RB6 high ...

RB7 high (input BAD) ...

Start over ... (^C to exit)

VPP off, VDD off, RB6 low, RB7 low (input OK) ...
[root@jml pp-0.6]#
```

6 Et les utilisateurs de Windows ?

Je me suis laissé dire qu'il y avait des utilisateurs qui continuaient à utiliser Windows. Qu'ils se rassurent, la présente étude s'applique en principe aussi à leur cas. Gpasm est écrit de telle sorte à être compatible avec MPASM de MICROCHIP, logiciel qui tourne sous Windows. Celà signifie à contrario que MPASM est compatible avec gpasm ! MPASM peut être facilement téléchargé sur Internet, par exemple sur le site de MICROCHIP.
Comme je l'ai déjà exposé, la récupération de pp sous forme d'archive .tar fournit aussi le fichier PP.EXE qui tourne sous DOS. La grande toile fournit par ailleurs de nombreuses possibilités de récupérer ce fichier, par exemple sur

`http://www.vermontficks.org/picpgmr.htm`

Il semblerait, selon cette page, que les utilisateurs de W2k doivent s'attendre à des soucis.... Dans le monde LINUX en tout cas j'ai essayé avec MANDRAKE 8.2 et RED-HAT 7.0 avec le même succès.
Il ne devrait donc y avoir aucune difficulté majeure pour les éventuels utilisateurs de Windows

à exploiter les informations consignées ici, encore que je ne puisse nullement vous assurer d'un fonctionnement correct (Je ne travaille que sous LINUX).

7 Conclusions

7.1 Un mot d'encouragement

On peut lire ici et là que le port parallèle d'un PC est un organe fragile, que sa destruction entraîne un dommage irréparable, surtout s'il est intégré à la carte mère. Bon, c'est juste mais en respectant quelques principes exposés plus haut, en particulier le montage des résistances dans le capot du connecteur je ne vois pas quel scénario pourrait conduire à la destruction de ce composant. Par ailleurs, si vous voulez vraiment faire preuve de prudence utilisez donc deux piles de 9V, l'une pour la tension de programmation, l'autre pour alimenter le régulateur 5V. De la sorte votre montage est totalement isolé du reste du monde et la masse est imposée par le port parallèle.
Concernant la résistance du PIC je peux simplement vous indiquer que le seul PIC que j'ai acheté m'a servi pour tous les essais et continue à clignoter gaiement à coté de mon PC.
J'ai essayé d'être très complet dans mon exposé. Celà conduit à un pavé de plus de 20 pages en version imprimée. J'espère que le volume ne rebutera personne. Par contre le lecteur devrait trouver ici, du moins c'est mon espoir, tout ce dont il a besoin pour un premier essai.

7.2 Et maintenant ?

Arrivé à ce stade se pose la question de savoir quelle suite donner à ce travail. Chacun fera comme bon lui semble.

7.2.1 Simplifier encore ?

On pourra par exemple essayer de simplifier encore, peut-être supprimer la zener 12V et les resistances associées et laisser monter la tension de programmation à 14V (5+9). Pour ma part je n'ai pas essayé. J'ai réussi à conduire tous mes essais avec un seul PIC qui va continuer à me rendre des services dans les prochains temps, je ne souhaite pas faire cet essai au risque de dépasser de quelques décimales ce que peut supporter MCLR.

7.2.2 Et les autres PIC's ?

Une autre voie serait d'explorer la possibilité de programmer d'autres PIC, par exmple de PIC16F877 [2]. Je viens de parcourir la data sheet, c'est simplement Bysance, 8k de RAM flash (mots de 14 bits), 256 octets d'EEPROM, 6 entrées/sorties analogiques 10 bits etc. le tout dans un boitier 40 pattes, donc avec des E/S logiques à foison !

[2] 16.20 Euros chez GO-TRONIC, catalogue 2002-2003, www.gotronic.fr

7.2.3 Acheter un "vrai" programmeur

J'ai remarqué lors de mes surfs que les logiciels de programmation qui tournent sous LINUX utilisent généralement le port parallèle. Je ne saurais donc trop recommander, dans l'hypothèse ou vous voudriez par la suite faire l'achat d'un vrai programmeur, de tenir compte de cette réalité et de choisir un modèle qui utilise ce port. Attention donc car la majorité des matériels proposés par les revendeurs utilisent le port série.

8 Remerciements

J'ai la bonne habitude de remercier, même si elles ne lisent pas ma langue, les personnes qui, grace aux informations qu'elles ont mis en place sur Internet, on permis que, partant de mes connaissances d'électroniques des années 70 et alors que je n'avais jamais entendu parler au préalable de microcontrôleurs, j'arrive en moins de 3 mois à mettre au point cet article.
Je distinguerais tout spécialement :

- BIGONOFF, cité plus haut et dont le cours de PIC est, à mon avis, une référence dans le domaine
- David TAIT, lui aussi cité plus haut et qui a beaucoup publié sur le thème du hardware "rapide et sale"
- Chris WILSON, l'auteur de pp (co-auteur avec David TAIT ?)

9 L'auteur

Jean-Marc LICHTLE, Ingénieur Arts et métiers, promotion CH73.
A l'époque la RAM était constituée de réseaux de tores magnétiques, un par bit, traversés chacun de trois fils qui permettaient les opérations de lecture et d'écriture. L'expression "1 kilo de RAM" avait alors une signification peut-être différente de celle qu'elle a actuellement ! Il fallait le volume d'une machine à laver pour ranger quelques kilos de mémoire sur un IBM 1130.
Les commentaires sur ce travail peuvent m'être adressés à l'adresse suivante :

jean-marc.lichtle@gadz.org

jml